Interface between SysML and Sequence Planner Language for Formal Verification

Sathyamyla Kanthabhabhajeya, Joakim Berglund, Petter Falkman, Bengt Lennartson

Department of Signals and System
Chalmers University of Technology
Horsalsvagen 9-11, Gothenburg
Sweden

Abstract. This paper presents a method and software for interfacing Systems Modeling Language (SysML) and Sequence Planner Language (SPL). Exchange of information between different software tools is of major interest for modern manufacturing industries from early design to final implementation. SysML, with its structure as a common platform, can then be interfaced with other domain-specific modeling tools to achieve information exchange. This paper presents a method to interface SysML with a recently introduced language for operation sequences called Sequence Planner Language (SPL). By this method, necessary information from behavioral constructs of SysML model are extracted and structured in SPL. This language, being a formal, graphical language,can be used to formally verify the system for any blocking states. An academic and an industrial model developed in SysML are tested using the interface implementation and the results show that information from SysML can be visualized in SPL and formally verified to have no blocking states.

Introduction

Automation in manufacturing industry is used in order to reduce human workload, reduce time consumption and improve the quality of products, by utilizing different software application tools. These tools help to read, evaluate, control and display parameters from different applications. The interaction between tools, the usage of selective tools, and the exchange of information between tools are evolving into a complex framework. Automation in working methodology or workflow of a manufacturing industry is in a growing phase. The mechanical and software developers of a manufacturing system need a flexible approach for exchanging information during the design stage. The ideal workflow during the initial stages of a design project, including both mechanical and software development, is illustrated in Figure 1 (Kanthabhabhajeya et al. 2012). A common platform between two branches is utilized for exchange of information. The purpose of this platform is to achieve timely information in virtual, database and documented environments.

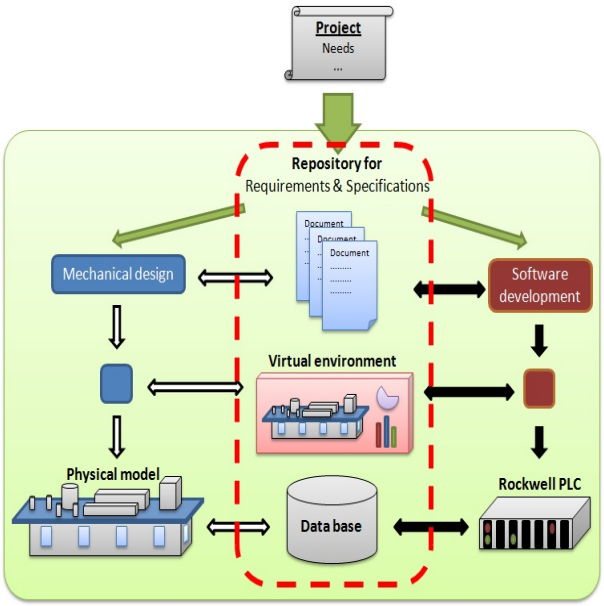


Figure 1: Ideal workflow of manufacturing industry (Kanthabhabhajeya et al. 2012)

The nature of the Systems Modeling Language (SysML) is to denote itself as a common platform and to interact with other tools and exchange information between developers. SysML, a graphical modeling language, serves as a semi-formal language with nine different diagrams (OMGSpecification 2010). In order to model a manufacturing system, these diagrams are used to describe the system from different perspectives. The structural and behavioral aspects of the model can be defined separately and add allocations to/from each other to interconnect these two main aspects. In-between these aspects is the requirement diagram which describes the context of the requirements of the system and this can be verified by referring the respective requirements to particulars of the model. The tool used in this paper for modeling in SysML is MagicDraw 17.0.1 beta 2 with a SysML plug-in developed by No Magic group (NoMagicGroup 2012).

Sequence Planner Language (SPL), (Lennartson et al. 2010) developed by the Automation group at Chalmers University of Technology, Sweden. SPL is a formal, graphical modeling language, used to define the manufacturing system with respect to its behavioral aspects focusing on operations and operation sequences. SPL being a formal language is mathematically defined over a set of tuples in extended finite automata (Section.Sequence Planner Language). The model in SPL is graphical and has formal definition that can be represented in a logical manner. SPL is also related to automaton language, described in detail (Section.Sequence Planner Language). The tool used for SPL is Sequence Planner (Bengtsson et al. 2011), developed by the developers team of the language itself. The SPL has a link to Supremica (a tool which is based on automata language), described in detail (Section.Sequence Planner Language).

In authors' previous work (Kanthabhabhajeya et al. 2012), SysML and SPL were compared by modeling an academic example and a Tetra Pak's TR28 Machine's filling module. These models in both languages are described in the following sections. Previous work shows both advantages and limitations of the two languages. SysML has an advantage that it is able to model both the structural and behavioral aspects of the system. The benefit of SPL is that it is a formal graphical modeling language, from which it is possible to perform both formal verification and control synthesis. This formal model can further be utilized for optimization of

process sequences. For modeling, SPL also provides a simpler solution for booking of shared resources for parallel operation sequences. Visualization with the tool Sequence Planner is more flexible than SysML's 'Views' and 'ViewPoints' as described in Kanthabhabhajeya et al. 2012. SysML adds the benefit of modeling structural aspects, it also adds the feasibility to model synchronous behavior of motors, especially in packaging machines. Along with this, it also has object flows in its activity diagram, which can even define objects that are of a 'continuous' nature, while SPL is limited to discrete behavior.

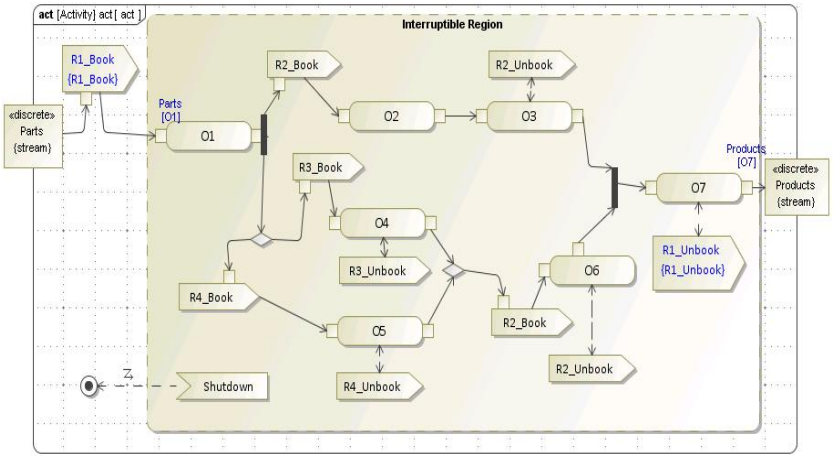
Following are the descriptions of similar contributions in the interface or exchange of information between SysML and different modeling tools. Johnson et al. 2008 proposed an approach to improve Model Based System Engineering (MBSE) by combining Modelica (a language for continuous dynamic models) and SysML. Another work on Multi-Representation Architecture (MRA) method was introduced along with simulation environments in SysML combined with CAD/CAE by Peak et al. 2007. Solutions for executable parametrics, reusable and modular solutions were then also produced.

Qamar et al. 2011 developed an infrastructure to support design of systems in a multi-domain framework and to achieve an integration with domain-specific models. The Mathwork's tool Simulink/SimMechanics was mapped with SysML and developed including an integration infrastructure. Kawahara et al. 2009 proposed an extension of SysML, including continuous-time behavior based on co-simulation of SysML and MATLAB/Simulink. A method was proposed by Evrot et al. 2007 and Petin et al. 2010 identifying safety properties by formal verification for systems modeled by SYSML. Any type of model checkers, like B-language (Turner et al. 2007), UPPAAL (Behrmann et al. 2006) and other DES model checkers could be used to formally verify the safety properties of the system and also mapped information and combined SysML with general formal models for verification of safety requirements. A verification method was presented by introducing Petri nets for formal behavioral modeling and Temporal Logic for formal verification along with SysML for the manufacturing system (Linhares et al. 2007).

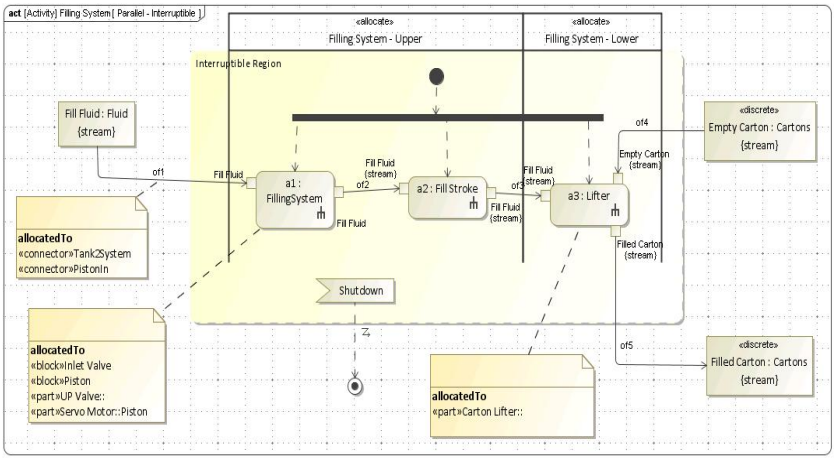
This paper proposes a way of getting the information from the model in SysML to SPL, and formally verifying the obtained model. In this paper, the implementation of the interface between SysML and SPL is developed, so the necessary information is identified and extracted from SysML and structured in the format of SPL. The SPL tool Sequence Planner along with Supremica are then used to perform formal verification. Apart from formal verification, SPL has an advantage over other formal languages like Petri nets with its visualization and graphical framework. This means that multiple perspectives of the sequences of SPL can be generated, such as product, resource, safety and human operator perspectives cf. Bengtsson et al. 2011. The structure of the paper is as follows, SysML models developed in previous work (Kanthabhabhajeya et al. 2012) are briefly described in the following section (SysML Models). Then, a short introduction to SPL is given (Sequence Planner Language). In Section Methods, which in detail lists the necessary information, mapping of information, extraction of data and structuring it in SPL. Succeeding section tributes the obtained results including formal verification by Supremica followed by Conclusion and future work.

SysML Models

An Academic example and Tetra Pak's Filling Module of TR28 were modeled in SysML to analyze the feasibility of modeling different systems (Kanthabhabhajeya et al. 2012). In this section, some particulars of the two example models are described.



(a) Activity diagram of Academic Example



(b) Activity diagram of TR28 Filling Module

Figure 2: SysML models - Behavioral part (Kanthabhabhajeya et al. 2012)

The activity diagram of the Academic example (Figure 2a), defines the sequences of actions that are performed. The model also includes activities that occur in parallel, shown as fork/join nodes and decision/merge nodes that express the choice between activities according to the condition for selection. It is assumed that the control flow in this model passes through all actions with equal priority. Other than the normal sequence of process activities, this academic model also include a constraint in the use of resources for certain activities. One example is that, resource R_2 has to be used by either actions O_2 and O_3 or by action O_6 as shown in Figure 2a. This resource booking/un-booking requires an additional modeling approach in SysML using SignalActions, which are expressed in a state machine diagram defining the SignalEvents and Triggers. The same type of booking procedure is used to achieve mutual exclusion between different activities. The state machine diagram and procedure for resource booking/un-booking are discussed in detail in (Kanthabhabhajeya et al. 2012).

The model, shown in Figure 2b describes the behavior of a Tetra Pak TR28 Filling Module. The other diagrams of the model defining in detail the structure and the object flow through the module are illustrated in previous work (Kanthabhabhajeya et al. 2012). The academic example along with TR28 model are used to validate the interface between SysML and SPL. The activity diagram of the TR28 Filling Module Figure 2b gives a brief description about the process flow. This model also recognizes the object flow, when the two objects, in this case the

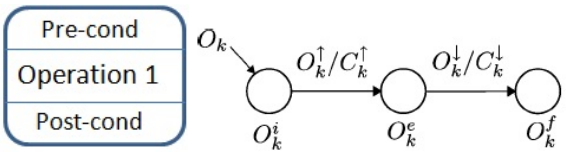


Figure 3: Operation with pre and post conditions in Sequence Planner.

fluid and cartons merge, as well as allocation of resources for the different actions.

Sequence Planner Language (SPL)

The formal graphical modeling language defined as Sequences of Operations (SOPs) (Lennartson et al. 2010) was titled Sequence Planner Language (SPL) by Kanthabhabhajeya et al. 2012. SPL defines sequences of operations that are necessary for the manufacturing system to achieve a product (Kanthabhabhajeya et al. 2012). A brief description of the language SPL is given in this section, using the academic example and Tetra Pak's TR28 filling module.

Operation, Resource, Liaison and View are the four major modeling elements in SPL, which possess formal definitions. In a manufacturing system, an operation is an activity performed by the system at a particular time and under specific conditions. An Operation in SPL is formally defined by a three state Extended Finite Automaton (EFA) (Skoldstam et al. 2007) where EFA is a generalization of an automaton including variables, guards and actions (Bengtsson et al. 2011).

Definition An extended finite automaton is a 7-tuple

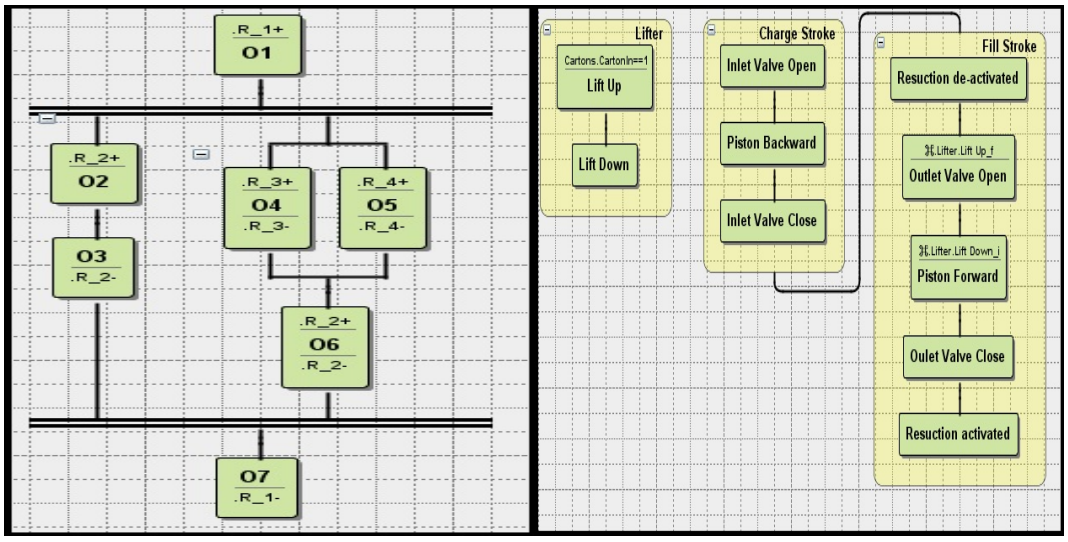
$$E = \langle Q \times V, \Sigma, \mathscr{G}, \mathscr{A}, \rightarrow, (q_0, v_0), M \rangle$$

where the set $Q \times V$ is the extended finite set of states, Q is a finite set of *locations* and V is the finite domain of definition of the variables, Σ is a nonempty finite set of events (the alphabet), $\mathscr{G}$ is a set of guard predicates over V, $\mathscr{A}$ is a collection of action functions, $\rightarrow \subseteq Q \times \Sigma \times \mathscr{G} \times \mathscr{A} \times Q$ is the set of state transition relations, $(q_0, v_0) \in Q \times V$ is the initial state, and $M \subseteq Q \times V$ is a set of marked desired states.

An operation is extended to its equivalent EFA with initial, executive and final locations (O_k^i, O_k^e, O_k^f) with guards and actions represented at the transitions as conditions on the current and updated value of the variables ($C_k \leftrightarrow \mathscr{G}/\mathscr{A}$).

Definition An operation is an EFA where the set of locations $Q_k = \{O_k^i, O_k^e, O_k^f\}$, the event set $\Sigma_k = \{O_k^{\uparrow}, O_k^{\downarrow}\}$, the set of transition conditions $C_k = \{C_k^{\uparrow}, C_k^{\downarrow}\}$, the transition relation $\rightarrow_k = \{\langle O_k^i, O_k^{\uparrow}/C_k^{\uparrow}, O_k^e \rangle, \langle O_k^e, O_k^{\downarrow}/C_k^{\downarrow}, O_k^f \rangle\}$, the initial location $q_k^i = O_k^i$, and all locations are marked, i.e. $M = Q$.

Both example models in SysML are modeled in SPL directly for comparison. The Academic model of SPL has a similar structure as the SysML model as shown in Figure 4a, whereas resource booking and un-booking is represented in SPL in a simpler way compared to the SysML model (Kanthabhabhajeya et al. 2012). In SPL, resource booking/un-booking can be defined in a model as pre/post-conditions in related operations. The fact that mutual



(a) Academic Example (b) TR28 Filling Module

Figure 4: SPL Models (Kanthabhabhajeya et al. 2012)

exclusion between resources is easily expressed and also helps the flexibility and simplicity of resource booking/unbooking. A detailed comparison between other modeling elements of SPL and SysML are made in the following sections.

The SPL model of the Tetra Pak filling module includes the three main operations *Lifter, Charge Stroke, Fill Stroke* performed by the machine. The SPL model also includes the detailed operations performed under these main operations to add clarity, which is not described in SysML. The details of the resources, allocation of blocks/resources, and other object flows are not explicitly modeled in SPL, while they are available in SysML. The operations in SPL, as shown in Figure 4b, carry pre-conditions which add constraints to the existing sequence to express the specification of the machine.

Methods

This section describes in detail the practical methods used for implementation of the interface between the SysML Activity diagram and SPL. Listing of information required, mapping this information towards SPL, methods for extracting data from SysML, and parsing the obtained information to SPL are discussed in this section.

Required Information

The purpose of formal verification in the context of this paper is to verify if there exists any blocking states or not in the process/sequence where the blocking states are defined as states that leads to a deadlock state. From the detailed SysML model, it is now necessary only to extract the behavior of the system. SysML, being a common environment for both Structural and Behavioral types of modeling to pursue formal verification, it is the behavioral modeling that is of relevance for SPL.

The process flow with different restrictions on actions performed by the system are required to create sequence of operations in SPL. The process flow for each action can be in a simple

sequence i.e one after the other, or sometimes with decision/merge nodes or fork/join nodes. The complete process flow with all nodes, along with constraints on the actions, the actions being *allocatedTo* or *allocatedFrom* any blocks or states of other diagrams, are also part of the information that is required to be extracted from the SysML model. Along with this it is essential to know the control and object flows between the actions, and if there are any other types of diagrams allocated for particular actions. If there are any state machine diagrams linked to actions of the current activity diagram, it is required that the states and transitions of that state machine diagram are perceived. Guards and actions of each transition in state machine diagrams are to be required whenever, an event is triggered from the actions of the activity diagram. It is usually the Call Behavior Action that represents the different performances of the system. Sometimes it is also different types of actions that are involved, like 'Send/Receive Signal Actions' in the activity diagrams. In this work, information from the activity diagram has been restricted to 'Call Behavior Actions' and 'Signal Actions' for simplification and these are the ones that are of significance. The information listed in this section are mapped with SPL elements in the following section.

Mapping of Information

Having listed the information, it is appropriate to map the obtained data with respect to the SPL format to be utilized for verification and other purposes. Table 1. maps the information between SysML and SPL.

Table 1: Table 1. Information Map between SysML and SPL

SysML Model	SPL Model	Comments
Activity Diagram	Operation View	
Actions	Operations	
Object/Control flow	Sequence of Operations	Both object and control flows in the same path in SPL
Decision/Merge node	Alternative Operations	
Fork/Join nodes	Parallel Operations	
Send/Receive Signal Actions	Pre-condition/Post-condition	It includes both guard and action
Constraints	Pre-condition/Post-condition	
AllocateTo or From	Realized By	Each operation can be realized by Resources

In Sequence Planning Language (SPL) section, each Operation itself includes its behavior with its neighboring Operations. The logical equations representing these relations and transitions will be difficult to visualize for the complete sequence. Hence, an Operation View is the environment where each Operation has its location and the graphical view of sequences between the operations is portrayed. Any number of operation views can be created, to define different sequences of operations for the same project. The Operation View of SPL can be directly mapped to the Activity Diagram of SysML. An Operation of SPL and an Action of SysML are models that intend to define a behavior, or activity, performed by the system.

In SPL, object flow and control flow, flows together along with the sequence of operations. An alternative relation in SPL is logically expressed as $O_k + O_l$. This means that if O_k OR O_l is not in its initial location of the Operation, then other operation cannot start (Bengtsson 2012).

This operation is similar to a Decision/Merge node of SysML where a decision is made for execution of one operation, with respect to a defined condition becoming true. Parallel relation of SPL, expressed as $O_k||O_l$, indicates that both O_k AND O_l are executed by any combination of events that exist in this order, similar to Fork/Join nodes in SysML.

Other constraints of the SysML model can be represented with pre/post-conditions in respective Operations of the SPL model. These conditions are used to express booking/unbooking of Resources but can also be variables with any type of logical conditions. Signal actions in the SysML model are also possible to model in SPL with pre/post-conditions. AllocateTo/From usually refer Blocks or other States that are allocated with Actions in the SysML model. This behavior is coordinated with Realization of Resources to respective Operations in SPL. State Machine diagrams allocated to some of the actions are represented using the logical description of pre/post-conditions of particular Operations. The entire mapping of information is organized in a .sopx file which inherits an Extensive Markup Language (XML) format. This file format of SPL has a structure to describe the model. It begins with Resources, Liaisons, Views and Operations; and inside Operations the information on pre/post-conditions, operation relations and other conditions are described. This structure of .sopx file will be discussed in detail in the following sections.

Information Extraction

As mentioned in SysML Models section, the tool used for representing the model in SysML is MagicDraw UML with a SysML plug-in developed by No Magic group. Similar to many simulation tools, MagicDraw also provides access to the model built in its framework in two ways; one with an export function of the project to an XML Metadata Interchange (XMI) file according to SysML standard, the other way is to interact with the tool using its Application Programming Interface (API) and gather the necessary information from the project. The XMI file generated by exporting the project is huge containing all data that is defined in the model. The size of the file will need some Mega Bytes, and parsing the necessary information from this file will increase the search processing time and time for developing an efficient algorithm for performing this search. As the list of information (Required Information) is known and it is only related to the behavioral model, it is safe to utilize the API to extract information from the model.

Plug-in is the only way to access the functionalities or methods using the API of in MagicDraw. The plug-in directory, including the compiled java file, which utilizes the methods of the API and also defines new functions for MagicDraw and a descriptor file listing the requirements for the plug-in, interacts with MagicDraw and loads the plug-ins during the time of initialization. Practical usability and implementation details of plug-ins are described in detail in the manual provided by MagicDraw on "MagicDraw - OpenAPI User Guide". This guides the developer to utilize default classes that are necessary for MagicDraw API. There also exist different methods, described in the manual, to create plug-ins and access the model in MagicDraw according to the needs of the developer. In this paper, it is the Eclipse-framework that is used for interacting with the SysML API and to develop the algorithm for the interface.

Built-in methods from the MagicDraw API help to get certain information from the model. Information directly available from the model can be extracted using these built-in methods of the API while certain data needs user-defined methods in addition for extraction. A few of the built-in methods used are listed here;

getProject() - gets current Project and all the diagrams related to this project
getActiveDiagram() - the currently active diagram type (bdd, ibd, activity, etc.,)

getUsedModelElements() - get the elements used in this model one after the other

Extraction of information from the SysML model is not always straightforward. This type of information requires user-defined methods and algorithms in order to extract. A few notable algorithms used are described here;

Algorithm 1 Get Initial Node Method - getInitialNode(ActivityNode *oneElem*)

```
repeat
  input oneElem : Activity Node currentNode : Activity Node - updated in each search

  currentNode = oneElem

  if currentNode is outputPin then
    currentNode = action of outputPin
    continue
  end if

  if currentNode is Action AND currentNode has Input then
    outerLoop:
    for all inputPin of currentNode do
      if inputPin is ObjectFlow then
        currentNode = Source of inputPin
        if currentNode is not ActivityParameterNode then
          break outerLoop
        end if
      end if
    end for
  end if

  if currentNode has Incoming then
    for all ActivityEdge of Incoming do
      if ActivityEdge is ObjectFlow then
        currentNode = Source of Edge
        break
      end if
    end for
    continue
  end if
  break
until true
```

Algorithm 1 identifies the initial node of the current activity diagram. This algorithm (Algorithm 1) acquires a node while calling this function and begins its loop first to check if the input element (*currentNode*) itself is an initial node or check for other options that follow in the algorithm. There are three options to identify what the *currentNode* is and continue with the loop by tracing out the previously linked nodes to the *currentNode*. The first option is to verify if the *currentNode* is one of the output pins, then the second option is to check if it is an instance of an Action, and the third option is if it has any incoming edges. In each option it follows a set of procedures to identify the previous node and assign the previous node as

currentNode and continue with the loop. The loop breaks itself when the *currentNode* does not have any previous nodes in the activity diagram.

Algorithm 2 generates the resource booked/un-booked information. The Signal Actions are checked for the respective Action node. The resource booking or un-booking information is gathered from the Send Signal Actions together with the Event triggers in the state machine diagram. In this algorithm (Algorithm 2) to get Booked Resource for a particular Action, the search begins with the Signal that has SignalEvent. The SignalEvent identified also has a TriggerEvent, which defines the resource that is booked. Another example of a useful method is the identification of the respective pairs of MergeNode and JoinNode.

Algorithm 2 Get Booked Resource Method - getBookedResource(SendSignalAction *currentNode*)

input *signal* : **Signal of** *currentNode*

if *signal* **is not empty then**
  if *signal* **has** *EventOfSignal* **then**
    signalEvent = **getSignalEvent of** *signal*
    if *signalEvent* **has** *TriggerOfEvent* **then**
      return getResourceID(*signalEvent*)
    end if
  end if
end if

Structuring in SPL

The information extracted from the SysML model is listed in an array with respect to the actual type of element. This has to be scripted in the structure, which can be accessed by SPL. The file format that SPL uses is XML format with a unique structure and saving with the extension .sopx. The Sequence Planner project file format includes five main type of elements, Global Properties, Resources, Liaisons, Views and Operations. If there exist any global properties in the project to be added, that will come under the first element. Liaisons define the interface between two entities in the same domain of the model. Views give the structure and graphical arrangement of the model in Sequence Planner. According to the pre/post-conditions of operations, respective operation relations are used (parallel, alternative) and represented in a sequence of operations. Views also helps to visualize the sequence of operations in different perspectives (Kanthabhabhajeya et al. 2012). In the two examples discussed in this paper, it is required to update with information on Resources and Operations. Visualization of this model in Sequence Planner is made by automatic generation of graphical sequences by the tool itself.

Information obtained from the extraction algorithms are fitted into the .sopx format. It is important to generate IDs for all elements and the ID needs to be a 4-digit numeric value (> 1005), the first five numbers are taken for the default main element types in SPL. One of the main algorithms for structuring the model in SPL format is discussed in this section.

Algorithm 3 checks for the elements of SPL in the array lists that are created while extracting the information from SysML. The algorithm will add each new element that is identified in the array list to the SPL structure. This algorithm also further checks for information regarding the particular element and adds description in SPL format. The algorithm adds operations in SPL, checks for any pre-conditions that are available, checks for any resource booking or

Algorithm 3 Add Operation Method - addOperation(ActivityNode *currentNode*, Array<Array<Integer» *preConditions*, Array<Integer> *bookResources*, int allocateResource)

```
input operationId : getOperationId(currentNode)

new operation Element
addContent(operation) to operationRoot
setAttribute to operation

new operationData Element
addContent(operationData) to operation
add default contents to operationData

if preConditions != 0 then
   new preSequenceConditions Element
   addContent(preSequenceConditions) to operationData

   for all list of preConditions do
      new or Element
      addContent(or) to preSequenceConditions

      for all value of list do
         new sequenceCondition Element
         addContent(sequenceCondition) to or
         setAttribute to sequenceCondition

      end for
   end for
end if

if bookResources is not Empty then
   similar steps followed for preConditions

end if
```

unbooking that is made and adds these details in the respective operation in a particular format. The algorithm works like adding a tree structure operation with its branches of pre and post-conditions and resource booking and other default elements.

Results and GUI

A Graphical User Interface is created in the MagicDraw environment to export the current activity diagram to the SPL project file. This GUI is added to the Help Menu of MagicDraw as shown in Figure 5. An activity diagram of the SysML model can be generated into a SPL project with the interface that has been implemented as shown in Figure 6b. It is feasible to add the algorithm as a plug-in to the MagicDraw resources and utilized for the interface with SPL.

The model of the Academic example in SysML, as shown in Figure 2a, is converted to a SPL Project file in .sopx format and opened in Sequence Planner. This model (Figure 6b) can

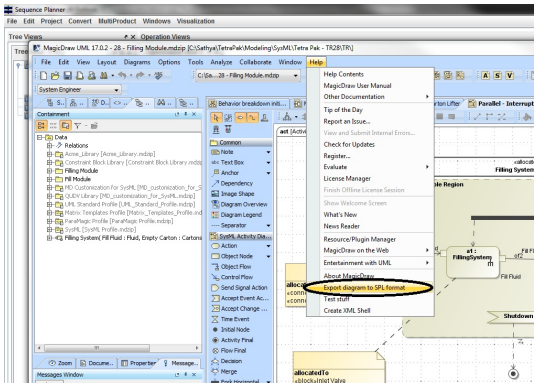
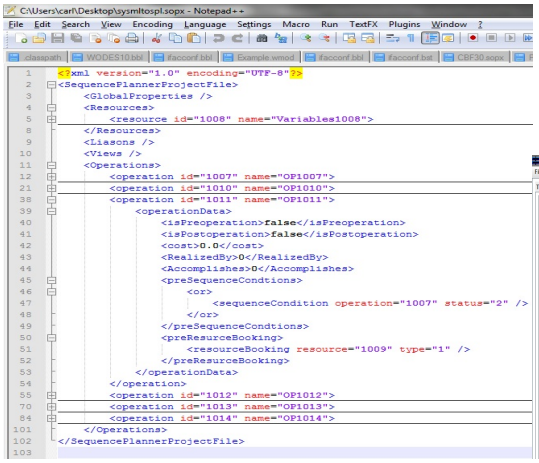
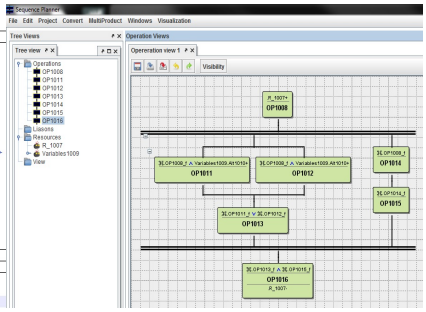


Figure 5: Graphical User Interface to generate SPL project from SysML Model

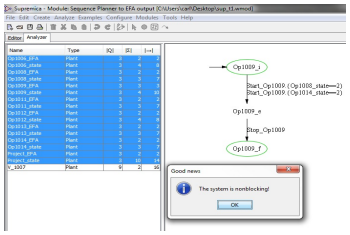
be compared to the Academic example in SysML (Figure 2a)and also to the one developed directly in SPL (Figure 4a). The resulting file format and the Sequence Planner model of this example is shown in Figure 6a. It has also been formally verified that the model generated in SPL from the SysML Activity diagram is 'non-blocking'. This formal verification with automata is performed in Supremica, as shown in Figure 6c. The resource booking/un-booking is made for resource $R1$ in operations $OP1008$ and $OP1016$ respectively. Other resources ($R2$, $R3$ and $R4$) in the SysML model do not include respective SignalEvents and Triggers being defined already. Hence, only the resource $R1$ is noticed in the SPL model. The extracted model of Tetra Pak's TR28 filling Module is as shown in Figure 7 in SPL. This is a simple sequence with three main operations obtained from SysML model (Figure 2b).



(a) SOPX File



(b) Academic Example in SPL



(c) Supremica: Non-Blocking

Figure 6: Academic Example Model in SPL and Supremica

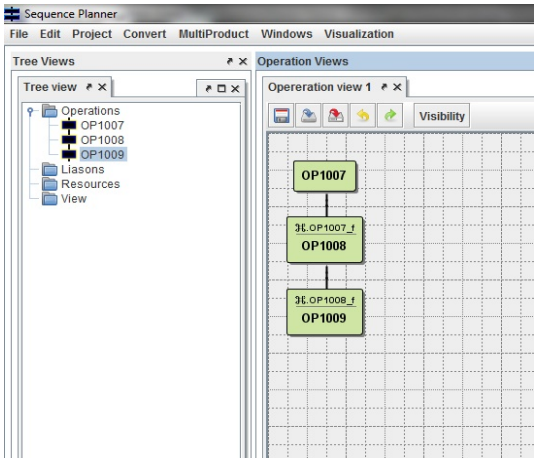


Figure 7: TR28 Fillling Module in SPL

Conclusions and Future Work

A system model with semi-formal representation such as SysML can clarify system properties by visualization and simulation. However, defining the sequence of operations for a system, specifying the resources for respective operation, and verifying a developed sequence across requirements, ask for a formal approach. Safety and nonblocking properties may then be formally verified, but optimal performance may also be achieved based on efficient optimization. This paper initiates the first step to interface SysML with a formal model called Sequence Planner Language (SPL) to extract necessary information from SysMLs' activity diagram and structure into the SPL format. The information generated in SPL from SysML can be formally verified by Supremica, which is based on automata models. Two models developed in SysML are tested with the algorithm developed for this interface and results are verified. The changes made in SPL inserted back in appropriate locations of the SysML model would be an area of interest as a continuation of this work.

Acknowledgement This work was carried out in collaboration with and support from Tetra Pak as well as the Wingquist Laboratory VINN Excellence Center within the Area of Advance, Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA). The support is gratefully acknowledged.

References

Behrmann G., et al. 2006. "Uppaal 4.0". *Proceedings of Third International Conference on Quantitative Evaluation of Systems*, pages 125–126.

Bengtsson K. 2012. "Flexible Design of Operation Behavior using Modeling and Visualization". PhD diss., Chalmers University of Technology (Sweden).

Bengtsson K., et al. 2011. "Sequence Planning using Multiple and Coordinated Sequences of Operations". *Accepted for publication in IEEE Transactions on Automation Science and Engineering*.

Evrot, D., et al. 2007. "Using SysML for Identification and Refinement of Machinery Safety Properties. *Proceedings of 1st IFAC Workshop on Dependable Control of Discrete Systems*, pages 127–132.

Johnson, T., Paredis, C., and Burkhart, R. 2008. "Integrating Models and Simulations of Continuous Dynamics into SysML. *In Proceedings of 6th International Modelica Conference*, pages 135–145.

Kanthabhabhajeya, S., Falkman, P., and Lennartson, B. 2012. "Systems Modeling Specification in SysML and Sequence Planner Language - Comparison Study". *14th IFAC Symposium on Information Control Problems in Manufacturing*, pages 1543–1550.

Kawahara, R. et al. 2009. "Verification of Embedded Systems Specification using Collaborative Simulation of SysML and Simulink Models". *Proceedings of 2nd International Conference on Model Based System Engineering*, pages 21–29.

Lennartson, B., et al. 2010. "Sequence Planning for Integrated Product, Process and Automation Design". *IEEE Transactions on Automatic Science and Engineering*, 7:791–802.

Linhares, M. V., et al. 2007. "Introducing the Modeling and Verification Process in SysML". *12th IEEE Internation Conference on Emerging Technologies and Factory Automation*, pages 344–351.

NoMagicGroup. 2012 "Magic Draw". Accessed 10 November. `http://www.nomagic.com/products/magicdraw.html`.

OMGSpecification. 2010. "OMG Systems Modeling Language (OMG SysML)". Accessed 10 November. `http://www.omg.org/spec/SysML/1.2/`.

Peak, R., et al. 2007. "Simulation-Based Design using SysML, Part 2: Celebrating Diversity by Example". *INCOSE*, LNCS 2102,.

Petin, J. F., et al. 2010. "Combining SysML and Formal Models for Safety Requirements Verification". *22nd International Conference on Software and Systems Engineering and their Applications*, pages 1–10.

Qamar, A., Wikander, J., and During, C. 2011. "A Mechatronic Design Infrastructure Integrating Heterogeneous Models". *Proceedings of International Conference on Mechatronics*, pages 212–217.

Skoldstam, M., Akesson, K., and Fabian, M. 2007. "Modelling of Discrete Event Systems using Finite Automata with Variables". *In Proc. 46th IEEE Conference on Decision and Control*.

Turner, E., et al. 2007. "Symmetry Reduced Model Checking for B" *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering.*, pages 25–34.

Biography

Sathyamyla Kanthabhabhajeya was born in 1985 in India. She received the B.E. degree in Electronics and Instrumentation Engineering from Annamalai University, India, in 2006 and worked as a Project Engineer in Sterlite Industries, India between 2006 and 2008. She recieved the M.Sc. degree in Systems, Control and Mechatronics from Chalmers University, Gothenburg, Sweden, in 2010. She has been working towards the Ph.D. degree in automation at Chalmers University of Technology, Sweden, since 2010. Her current research interests include systems engineering and modeling of manufacturing systems.

Joakim Berglund was born in Gothenburg, Sweden, in 1985. He finished his studies on the bachelor program Automation and Mechatronics in 2010 at Chalmers University of Technology, Gothenburg, Sweden. Since 2010, he has been working towards the M.S. degree in

systems, control and mechatronics at Chalmers University of Technology. On the way to the M.S. degree he has been working on a thesis related to industrial automation, supported by Tetra Pak, a global packaging company.

Petter Falkman was born in Gothenburg, Sweden, in 1972. He received the Ph.D. degree in Electrical Engineering in 2005 from Cahlmers University of Technology, Göteborg, Sweden. He is currently a Senior Lecturer at the Department of Signals and Systems, Chalmers University of Technology. Dr Falkman is currently program director for the Automation and Mechatronics program at Chalmers. He is also part of the Wingquist Laboratory, a research center for virtual product and production development at Chalmers. His main research topic concerns information integration, virtual preparation, and optimization of production systems.

Bengt Lennartson received the Ph.D. degree in automatic control from Chalmers University of Technology, Gothenburg, Sweden, in 1986. Since 1999, he has been a Professor of the Chair of Automation, Department of Signals and Systems. He was Dean of Education at Chalmers from 2004 to 2007, and currently he is Co-Chair of the RAS-TC on Sustainable Production Automation and Associate Editor for IEEE Transaction on Automation Science and Engineering. He is (co)author of 220 peer reviewed international papers with more than 3500 citations (GS). His main areas of interest include discrete event and hybrid systems, as well as robust feedback control.